
Subprograme C++ Clase de variable

Variabile C++

Variabilele dintr-un program C++ pot fi clasificate în:

1. Variabile globale 2. Variabile locale 3. Parametri formali

1. *Variabile globale*

- ▶ Se declară **în afara oricărei funcții** din program
- ▶ Sunt alocate static, în segmentul de date al programului
- ▶ Sunt inițializate implicit, cu valoarea 0
- ▶ Au domeniul de vizibilitate tot fișierul sursă, adică pot fi folosite din locul în care au fost definite și până la sfârșitul fișierului
- ▶ Au alocat spațiu în memorie tot timpul rulării programului



Variabile locale

2. Variabile locale

- ▶ Se declară **doar** în interiorul unei funcții din program, inclusiv în funcția main ()
- ▶ Sunt alocate implicit pe stiva sistemului
- ▶ **Nu** sunt inițializate implicit, dacă nu sunt inițializate explicit de programator, rețin o valoare oarecare, numită *valoare reziduală*
- ▶ Au domeniul de vizibilitate la nivelul blocului în care au fost declarate, adică pot fi folosite **doar** în cadrul acelui bloc de instrucțiuni
- ▶ Au alocat spațiu în memorie numai în timpul rulării blocului respectiv de instrucțiuni



Ex1

```
# include <iostream>
using namespace std;
int N;
void fl ()
{ int x=5; N=10;
  cout << endl<<N;
  cout << endl << x; }
void main () {
  N=4; cout << N;
  fl(); }
```

Ex2

```
# include <iostream>
using namespace std;
int N;
void fl () {
  int x=5;
  cout << endl << x;
  P=2 ; }
int P=9;
void main () { fl();
  cout << x;   P=7; }
```

Ex3

```
# include <iostream>
using namespace std;
int N;
void fl(int p)
{ int x=p;
  cout << x; }
void main ()
{ fl(3); }
```



Regula de omonimie

- În cazul în care există o variabilă locală care are același nume cu o variabilă globală, aceste două variabile se numesc variabile omonime.
- Variabilele locale sunt prioritare (ascund) variabilele globale omonime.

```
int n=10;  
void f1() { int n=2; cout << n; }  
void main () { f1();    cout << n; }
```



Întrebare. Cum gestionează compilatorul cele două variabile omonime ?

Răspuns:

- Variabilelor globale li se rezervă spațiu de memorare la începutul execuției programului, într-o zonă de memorie numită “zonă de date”. Acest spațiu va fi ocupat până la încheierea execuției programului.
- Variabilelor locale li se rezervă spațiu într-o zonă specială de memorie numită “stiva”. La încheierea execuției subprogramului, conținutul stivei este eliberat. Din acest motiv, variabilele globale sunt vizibile doar în interiorul subprogramului în care au fost declarate.



Parametri formali

3. Parametri formali – reprezintă o cale de comunicare între modulul apelant și funcția apelată. Pot fi:

- ▶ **Parametri de intrare** – corespund *datelor de intrare* din analiza problemei
 - Sunt valori transmise *de modulul apelant către funcția apelată*
 - Se transmit *prin valoare*
 - Se declară ca orice variabilă prin `tip idvar`
- ▶ **Parametri de ieșire (rezultate)** – corespund datelor de ieșire din analiza problemei
 - Sunt valori *transmise de funcția apelată către modulul apelant* - Se transmit *prin referință*, este specificat prin `tip& idvar`



- ▶ **Parametri de intrare și ieșire** – sunt parametri formali transmiși prin adresă, dar care sunt folosiți și pentru a transmite date de intrare

Parametrii formali și actuali

- ▶ Parametrii care sunt declarați în antetul unei funcții se numesc parametri formali, iar cei care se găsesc în instrucțiunea de apel se numesc parametri efectivi (actuali sau de apel).
- ▶ Legătura între *parametrii formali* și cei *actuali* este dată de regulile următoare:
- ▶ *Parametrii actuali **trebuie** să coincidă ca număr, tip și ordine cu parametri formali*



-
- ▶ *Transmiterea parametrilor are efectul unei atribuirii a valorii parametrului actual către parametrul formal corepsunzător*

Metode de transfer al parametrilor

- ▶ **Transmiterea prin valoare** se folosește atunci când funcția primește acea valoare ca o dată de intrare, fără a transmite în modulul apelant valoarea modificată în subprogram.
- ▶ Pot fi transmise prin valoare:
 - ▶ a) valorile reținute de variabile
 - ▶ b) valoarea unei expresii, care pot conține inclusiv apeluri de funcții; expresiile sunt evaluate înainte de transfer



- ▶ **Transmiterea prin referință** se folosește atunci când în urma apelului dorim ca variabila transmisă să rețină valoarea stabilită în timpul execuției subprogramului

Apelul unei funcții și revenirea din apel

- ▶ **Apelul unei funcții – suspendă execuția modulului apelant, până la revenirea din apel**
- ▶ a) reprezintă o *instrucțiune* pentru funcțiile de tip void
Numef (parametri_actuali);
- ▶ b) reprezintă *valoarea returnată de instrucțiunea return* pentru funcțiile cu tip, care poate fi inclusă în evaluarea unei expresii matematice sau într-o instrucțiune de afișare
- ▶ c) se salvează pe stiva sistemului înregistrarea de activare (I.A.) a subprogramului
- ▶ **Revenirea din apelul funcției**
- ▶ a) programul continuă *cu următoarea instrucțiune* pentru funcțiile de tip void

-
- ▶ b) pentru funcțiile cu tip se folosește valoarea returnată în evaluarea expresiei în interiorul căreia a fost apelată sau în instrucțiunea de afișare
 - ▶ c) se eliberează stiva sistemului memoria ocupată de înregistrarea de activare I.A. a subprogramului și se continuă executarea programului de la adresa de revenire A.R.

Înregistrarea de activare (I.A.)

- ▶ **Înregistrarea de activare I.A.** a apelului unui subprogram reprezintă toate datele salvate pe stiva sistemului la apelul subprogramului:
- ▶ *a) adresa de revenire (A.R.) este adresa instrucțiunii care se efectuează după apel și se alocă spațiu pentru rezultatul funcțiilor cu tip*



- *b) valorile variabilelor transmise prin valoare sau adresele valorilor transmise prin referință ► c) se alocă variabilele locale*

I.A.

- A.R. – adresa de revenire

- Spațiu pentru rezultatul funcției cu tip

-Valoarea parametrilor actuali transmiși prin valoare

-Adresa parametrilor actuali transmiși prin referință -

Variabilele locale